INFORMATION TECHNOLOGY LABORATORY

Performance of Service-Discovery Architectures in Response to Node Failures

Chris Dabrowski, Kevin Mills, Andrew Rukhin

2003 International Conference on
Software Engineering Research and Practice
(SERP'03)
Las Vegas, Nevada

June 23-26, 2003

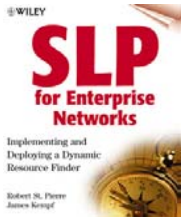
NIST
National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

Dynamic discovery protocols…

enable ***distributed software components***

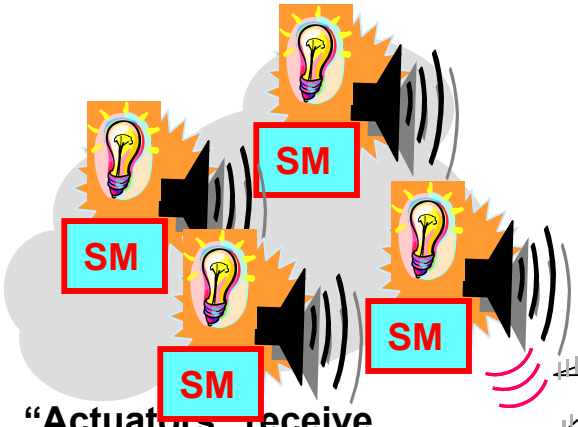
(1) to ***discover*** each other without prior arrangement,
(2) to ***express*** opportunities for collaboration,
(3) to ***compose*** themselves into larger collections that cooperate to meet an application need, and
(4) to ***detect and adapt*** to failures.

Some examples:

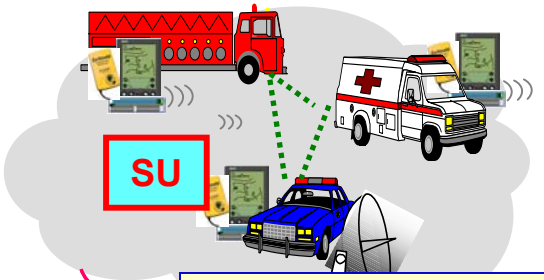
JINI **3-Party Design**	UPnP FORUM **2-Party Design**	 **Adaptive 2/3-Party Design**
The Salutation Consortium **Vertically Integrated 3-Party Design**	HAVi **Network-Dependent 3-Party Design**	Bluetooth™ **Network-Dependent 2-Party Design**

Service Discovery Protocols in Distributed Environments

Enable dynamic location and combination of remote services to perform critical, real-time tasks

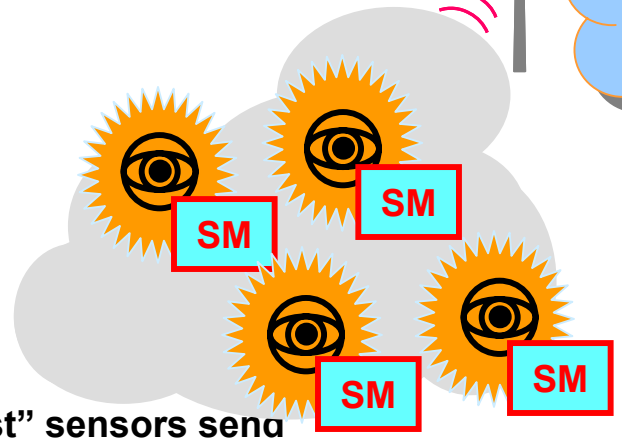


"Actuators" receive ad-hoc requests

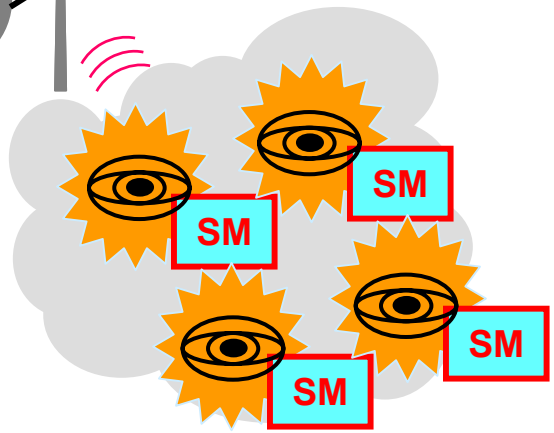


Mobile command post Service User dynamically locates and combines sensors and actuators

Core Network



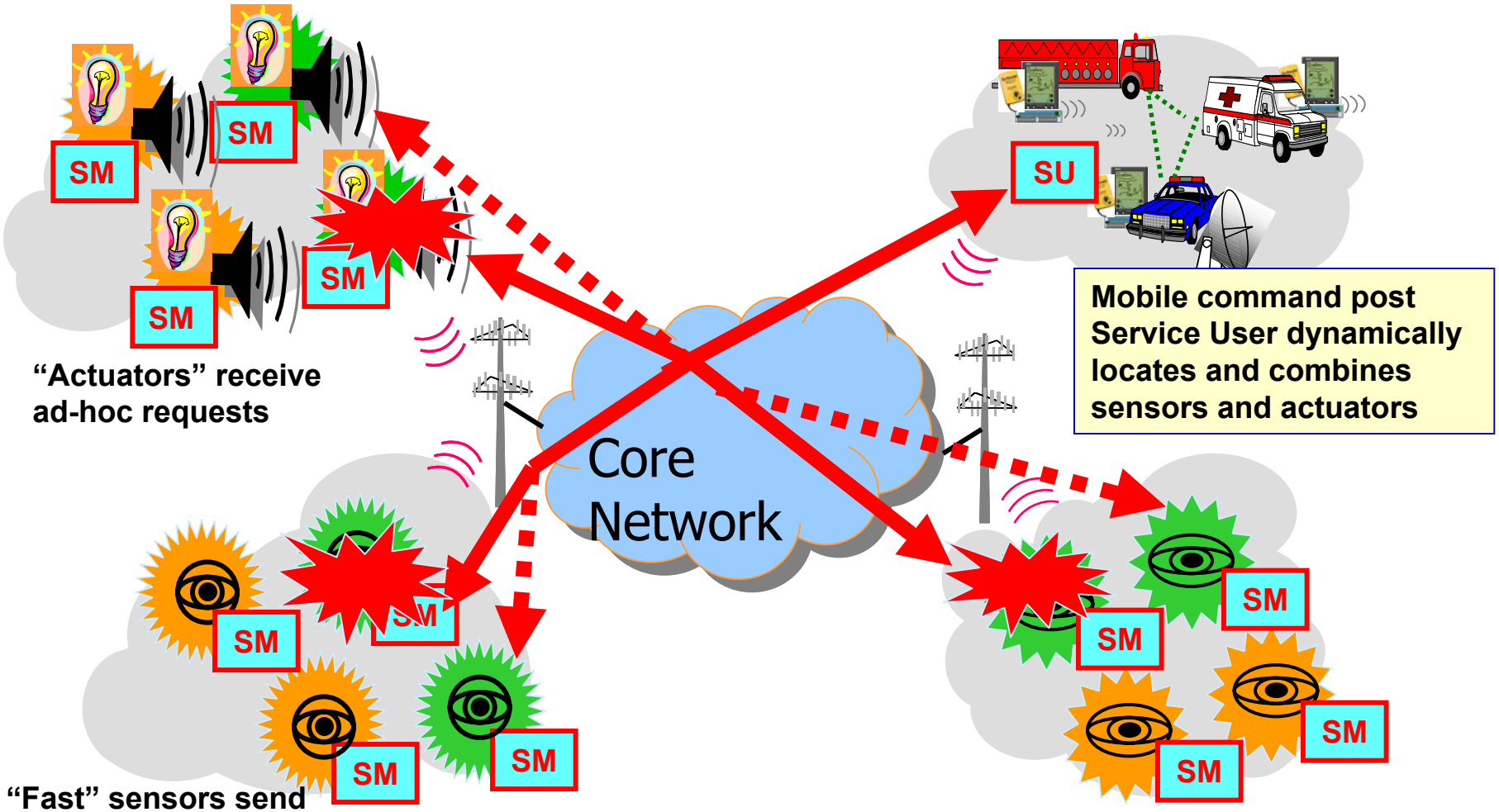
"Fast" sensors send readings every 2 seconds



"Slow" sensors send readings every 30 seconds

- SU is Service User
- SM is Service Manager

How Well Do Service Discovery Protocols Replace Services Lost to Node Failure?



"Actuators" receive ad-hoc requests

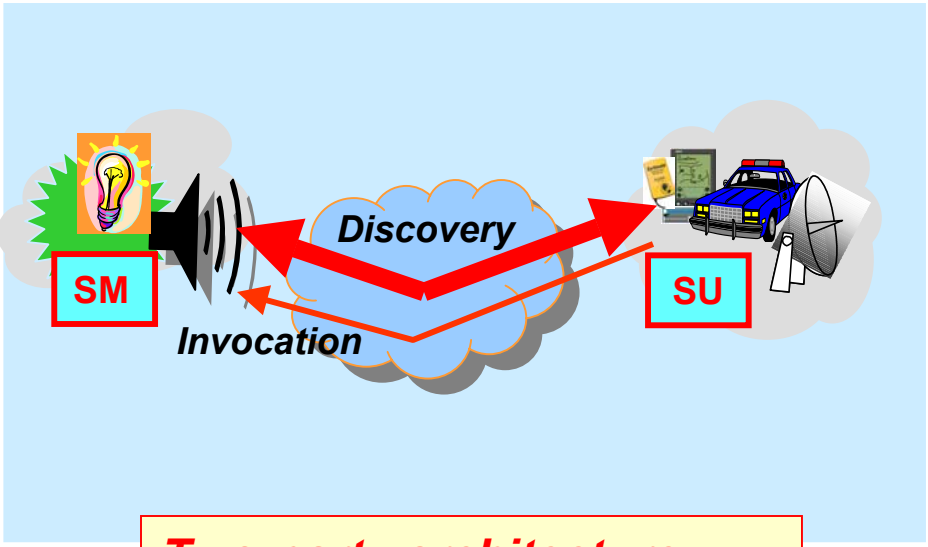
Mobile command post Service User dynamically locates and combines sensors and actuators

"Fast" sensors send readings every 2 seconds

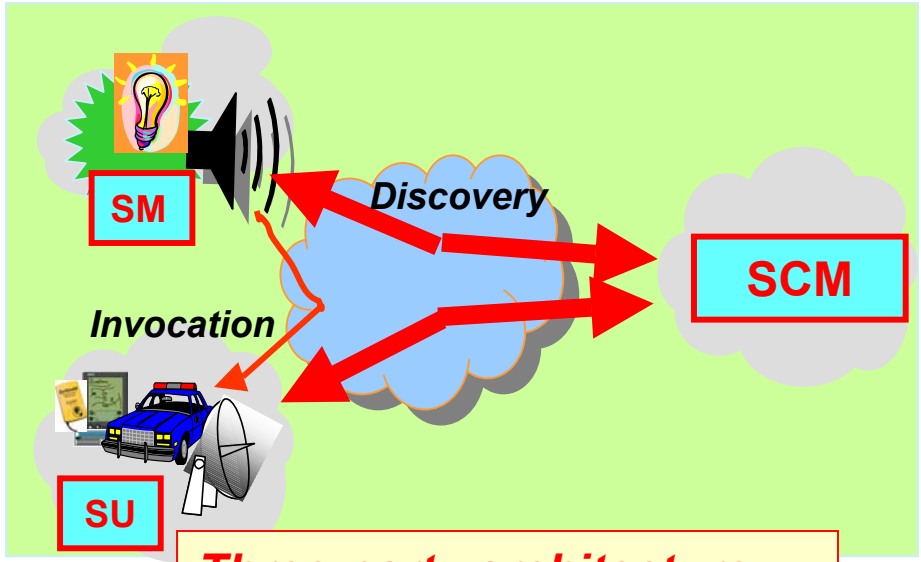
"Slow" sensors send readings every 30 seconds

- SU is Service User
- SM is Service Manager

Two generic architectures underlie most service discovery protocols



Two-party architecture

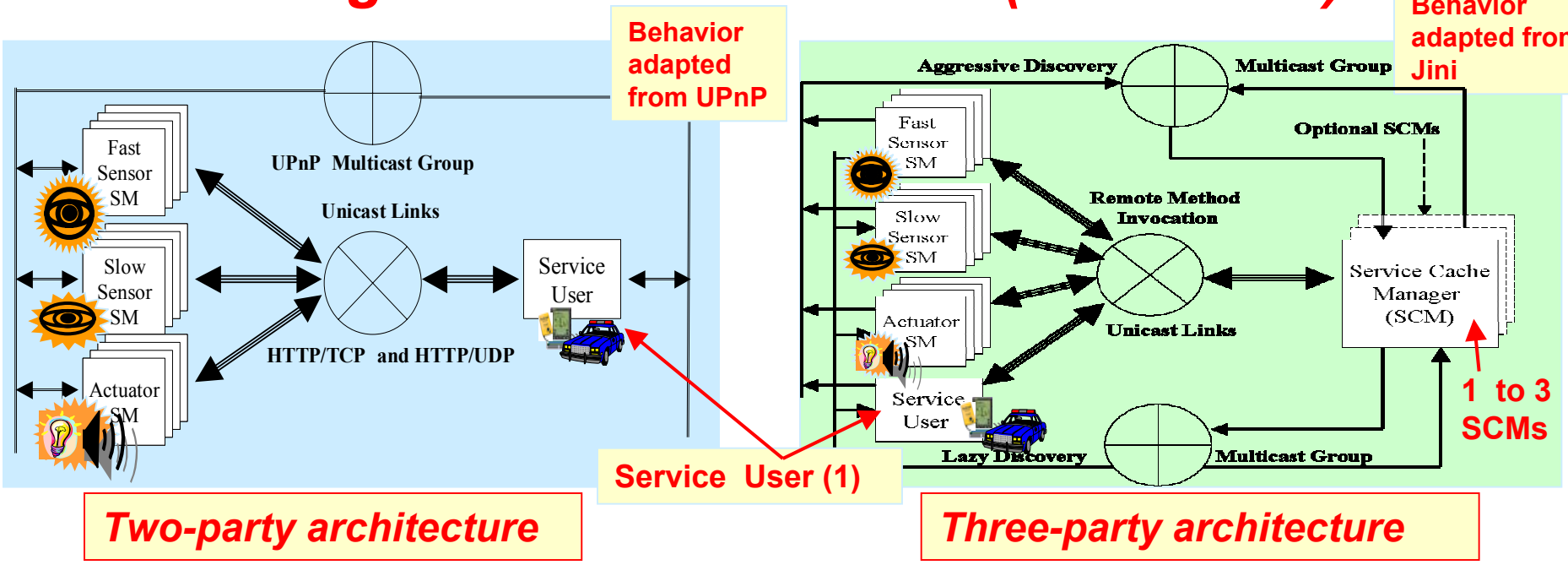


Three-party architecture

- SU is Service User
- SM is Service Manager
- SCM is Service Cache Manager

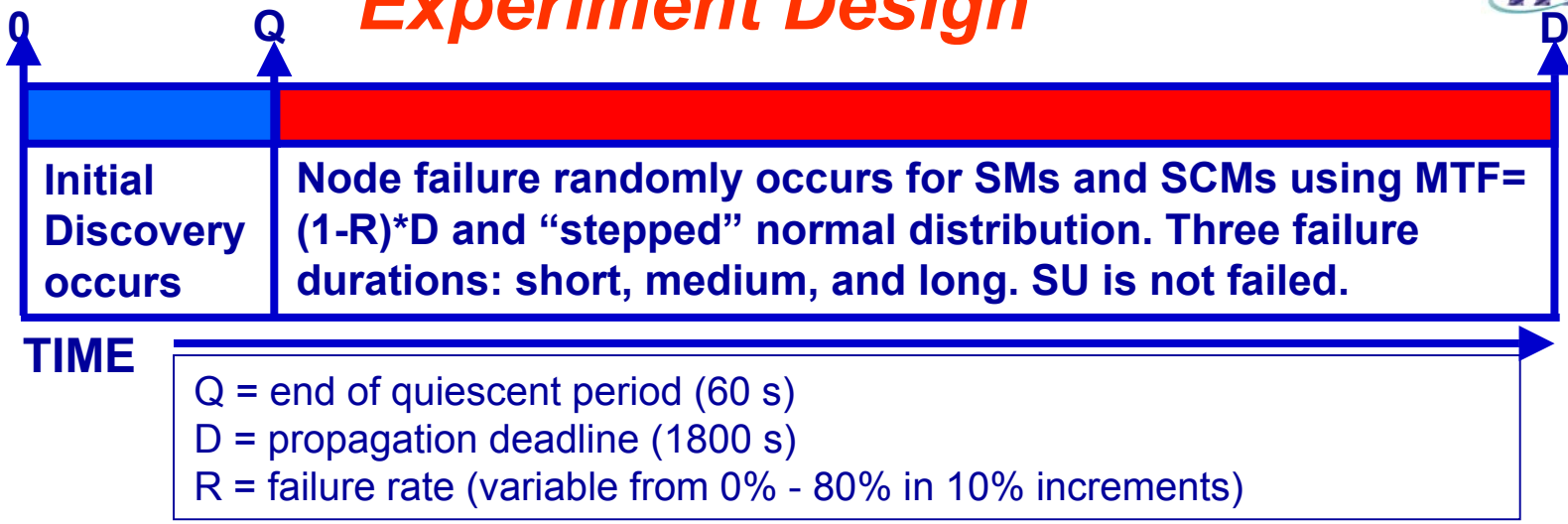
- **In two-party architectures**, Service Users discover Service Managers directly and invoke services
- **In three-party architectures,** both Service Managers and Service Users discover Service Cache Managers (SCMs); SU obtains services through SCM intermediary and then invokes

Two generic architectures (continued)



- **Discovery (2-party)**: SU discovers SMs through multicast search strategies
  - **Registration on SM**: SU registers for notification of change in service (renews every 300s)
- **Discovery (3-party)**: both SMs and SUs discover SCMs through multicast search
  - **Registration on SCM:** SMs register services (renews every 300s for fast sensors; 60s for slow sensors and actuators); SU registers notification requests (renews every 300s)
- **Failure Detection** by SU: through (1) SM non-response or (2) failure of registration renewal (heartbeat mechanism) and notification in 3-party case
  - **Recovery:**: 2-party SU multicasts queries to SMs every 120s
  - **Recovery:**: 3-party SU queries SCMs for service; If SCMs lost, SU listens for SCM announcements (every 120s) & SMs do the same

Experiment Design



- Goal of SU is to be ***functional***; e.g, to continually possess one instance of each type of service ("fast" sensor, "slow" sensor, & actuator).
  - When >= 1 type of sensor is missing, SU is ***non-functional***
  - To focus on alternative architectures & associated processes, mechanisms such as service caching factored out
- Formal conditions for measuring latency in detecting service failure and replacing lost service provide basis for metrics

Detecting Failure of Services in Use

Service User should hold services that are being actively managed (e.g. available)

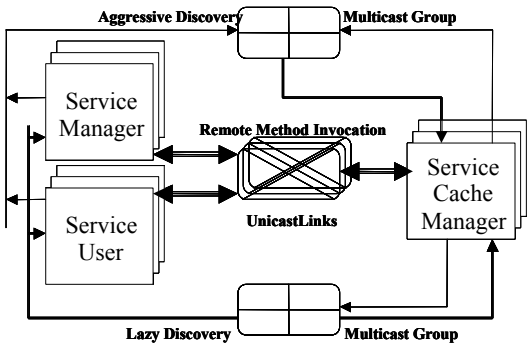
FOR All (SM, SU, SD)
(SM, SD) isElementOf SU discovered-services
implies
SD isElementOf SM managed-services

Recovering and replacing failed services

SDP should provide Service User with needed services if they are available

FOR All (SM, SU, SD)
SD [capabilities] isElementOf SM managed-services
SD [capabilities] isElementOf SU required-services
ResourceNeeded (SU, SD)
implies
(SM, SD) isElementOf SU discovered-services

Modeling and Analysis Approach: Use Rapide ADL to Model and Understand Dynamics of Service Discovery Protocols

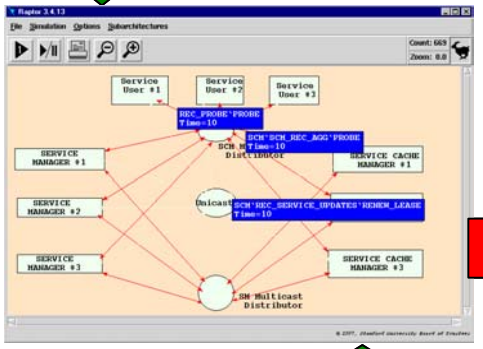


Topology

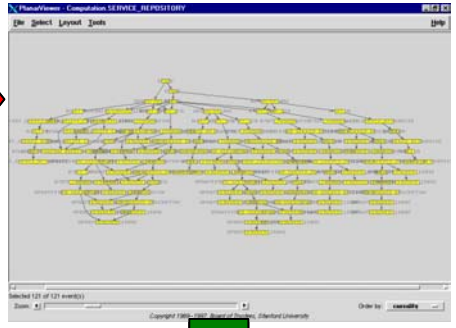
Scenario

Time	Command	Parameters
5	NodeFail	SM4
5	LinkFail	SCM1 SM4
10	GroupJoin	SM4 GROUP1
10	FindService	SU8 5 1 2 S XYZ ALL
50	AddService	SM4 SCM3 T ATT API GUI 20 30

Behavior Model



Execute with Rapide



```
-- ***************************************************
-- ** 3.3  DIRECTED DISCOVERY CLIENT INTERFACE     **
-- ***************************************************
-- This is used by all JINI entities in directed
-- discovery mode.  It is part of the SCM_Discovery
-- Module. Sends Unicast messages to  SCMs on list of
-- SCMS to be discovered until all SCMS are found.
-- Receives updates from SCM DB of discovered SCMs and
-- removes SCMs accordingly
-- NOTE: Failure and recovery behavior are not
-- yet defined and need reviw.
TYPE Directed_Discovery_Client
  (SourceID : IP_Address; InSCMsToDiscover : SCMList; StartOption : DD_Code;
   InRequestInterval : TimeUnit; MaxNumTries : integer; InPV : ProtocolVersion)
IS INTERFACE
SERVICE DDC_SEND_DIR    : DIRECTED_2_STEP_PROTOCOL;
SERVICE DISC_MODES      : dual SCM_DISCOVERY_MODES;
SERVICE DD_SCM_Update   : DD_SCM_Update;
SERVICE SCM_Update      : SCM_Update;
SERVICE DB_Update       : dual DB_Update;
SERVICE NODE_FAILURES: NODE_FAILURES; -- events for failure and recovery.
ACTION
 IN Send_Requests(),
    BeginDirectedDiscovery();
BEHAVIOR
   action animation_Iam (name: string);
   MySourceID        : VAR IP_Address;
   PV                : VAR ProtocolVersion;
```

Consistency Conditions

For All (SM, SD, SCM):
(SM, SD) IsEleme ntOf SCM registered -services (CC1)
implies SCM IsElementOf SM discovered -SCMs

For All (SM, SD, SCM):
SCM IsEl ementOf SM discovered -SCMs & (CC2)
(SD) IsElementOf SM managed -services
implies (SM, SD) IsElementO f SCM registered -services

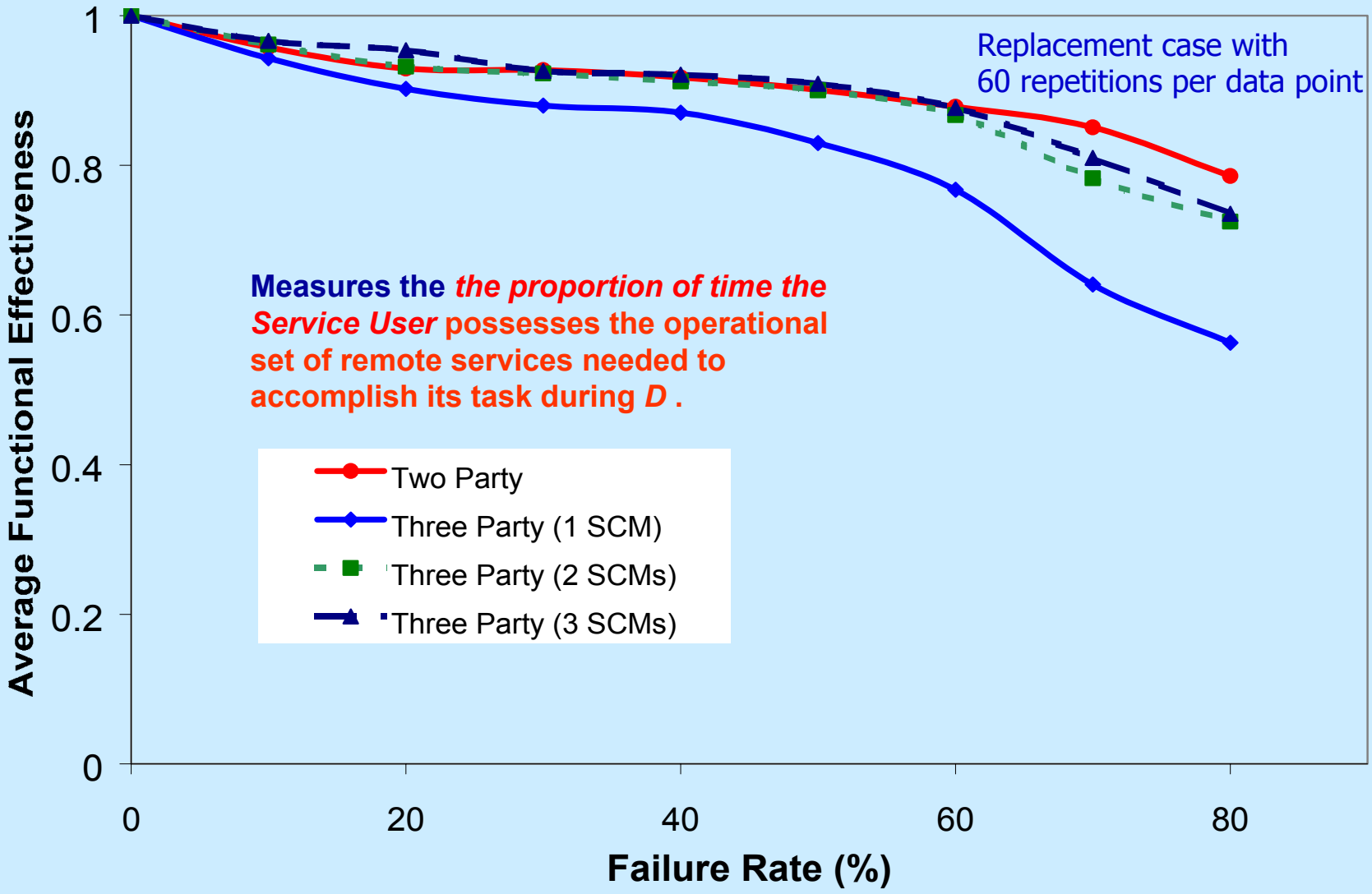
For All (SM, SD, SCM):
SCM IsElementOf SM discovered -SCMs & (CC3)
(SM, SD) IsElementOf SCM registered -services &
NOT (SCM IsElementOf SM persistent -list)
implies Intersection (SM GroupsToJoin, SCM GroupsMemberOf)

For All (SM, SD, SCM, SU, NR):
(SU, NR) IsElementOf SCM requested -notifications & (CC4)
(SM, SD) IsElement Of SCM registered -services &
Matches((SM, SD), (SU,NR))
implies (SM, SD) IsElementOf SU matched -services

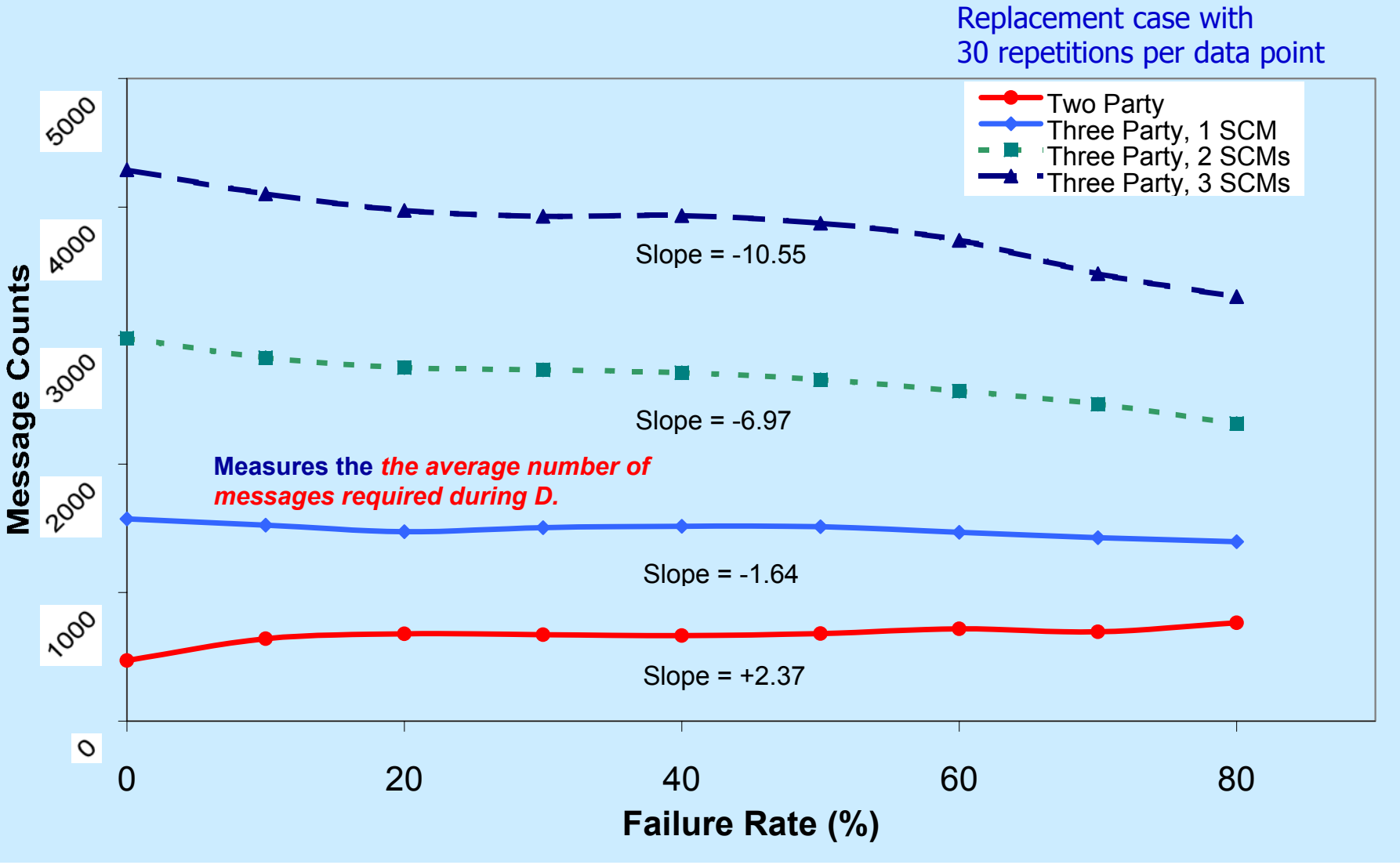
Analyze POSETs

Use metrics to Assess Correctness & Performance

Functional Effectiveness of Two-Party vs. Three-Party When One SM of Each Type is Always Available



Efficiency of Two-Party vs. Three-Party When One SM of Each Type is Always Available

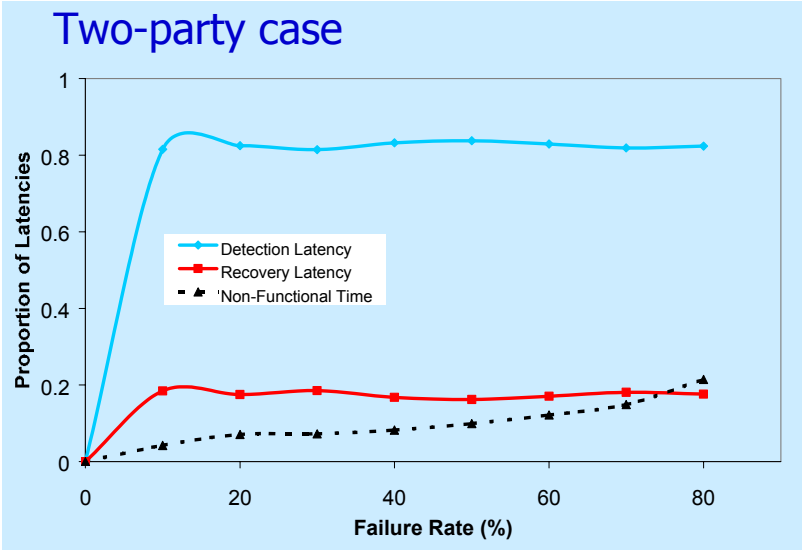


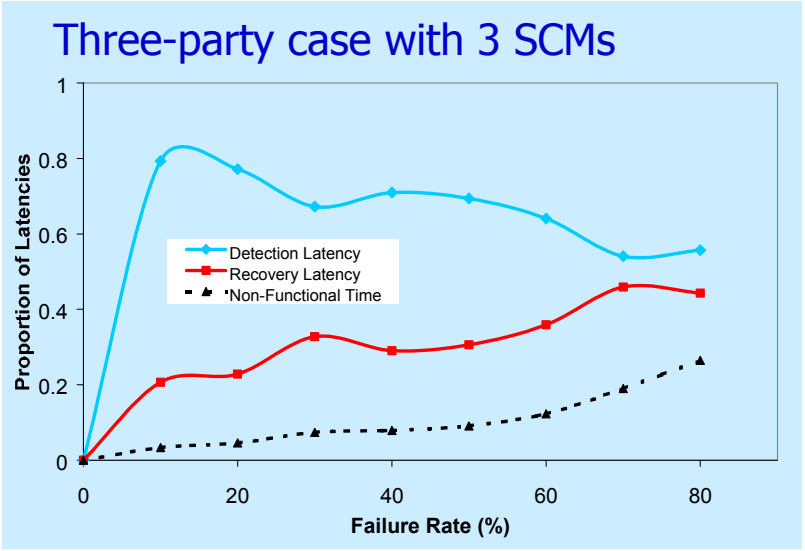
Detection and Recovery Latencies for Two-Party vs. Three-Party When One SM of Each Type is Always Available

Decomposing non-functional time:

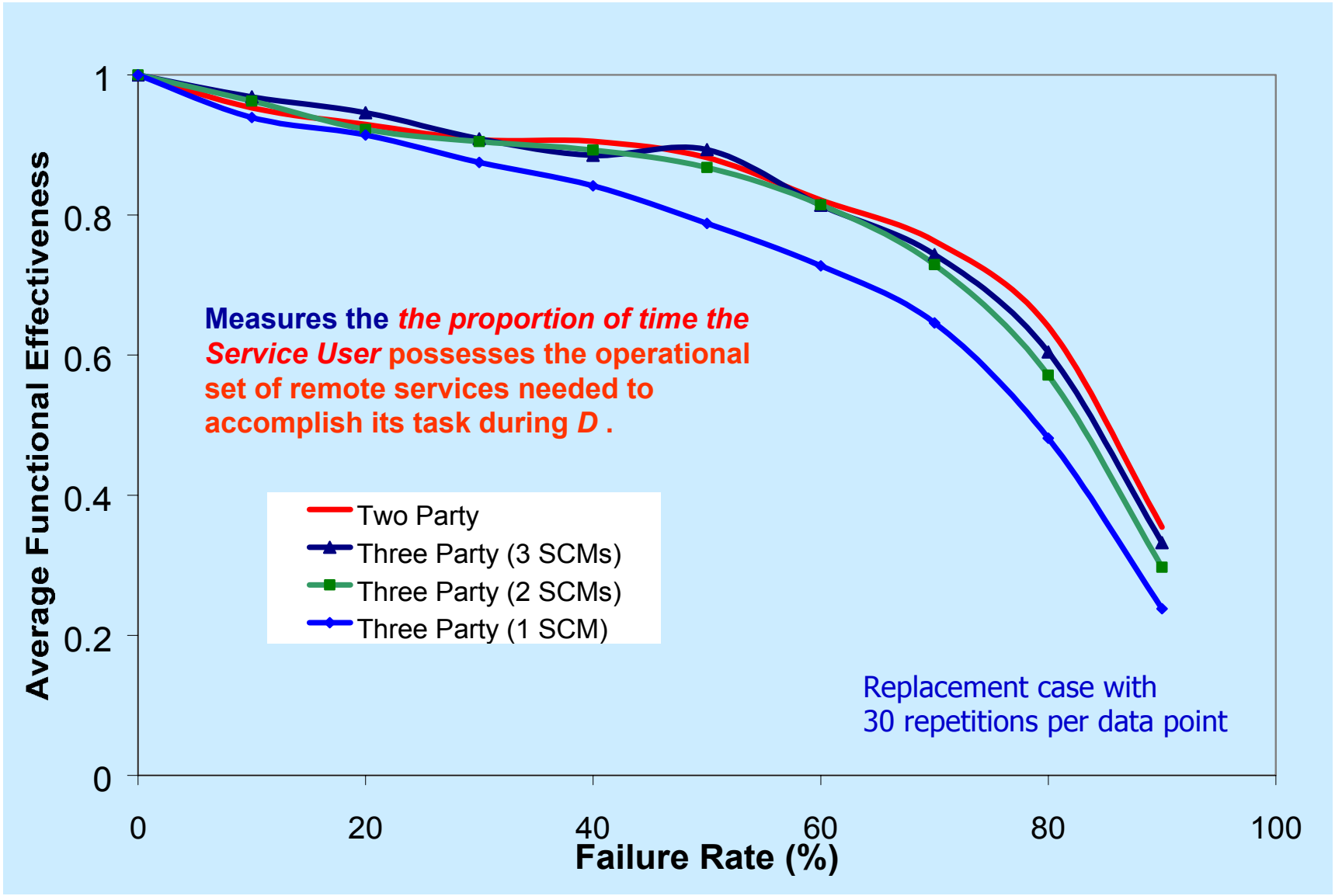
- **Detection Latency** - delay in detection failure
- **Recovery Latency** – delay in restoring required services

-> Detection latency was dominant in 2-party case; in 3-party case, proportion of recovery latency increased as failure rate increased due to unavailability of SCMs





Functional Effectiveness of Two-Party vs. Three-Party When All SMs of Each Type Can Fail



Conclusions and Future Work

- Service discovery protocols possess basic capabilities to enable failure detection and recovery under conditions of node failure.
- Results of experiments in node failure:
  - Three-party SCM is potential point of vulnerability at very high failure rates; reduced functional effectiveness
  - Effectiveness of three-party architecture approached level of two-party architecture as number of SCMs were added
  - Two-party architecture showed better efficiency than three-party architecture; redundant SCMs increases overhead (though subject to protocol variations in messaging)
  - Performance of both architectures can be improved by optimizing heartbeat mechanisms (registration refresh rate)
- Ongoing and Future Work
  - Repeat experiments with adaptable 3-party architecture that switches to 2-party mode when no SCMs can be found (SLP)
  - Investigate robustness of service discovery strategies in larger scale environments.